Errors in numerical analysis

Numerical analysis - University of Luxembourg – Session 2

Interpolation & Approximation - Exercices

Exercise 1. **Basic of Lagrange interpolation**

Write and plot the Lagrange basis associated to the points $\{x_0, x_1, x_2\} = \{-1, 3, 4\}$.

Then compute the Polynomial interpolant for the data $\{y_0, y_1, y_2\} = \{2, -3, 1\}$.

Compare with the built-in scipy.interpolate.lagrange function.

Finally plot on the same graph the interpolation points, the built-in interpolant and your implemented interpolant.

Exercise 2. **Interpolating Runge function**

Using the built-in scipy.interpolateLagrange interpolation, compute the absolute interpolation error by sampling n equidistant points for the function

$$g(x) = sin(5x), x \in [0, \pi].$$

Compare and discuss the error with the theoretical upper bound.

Repeat the exercise for the Runge function

$$f(x) = \frac{1}{1 + 25x^2}, x \in [-1,1],$$

for different values of n. Explain your observations and give a theoretical argument from the lecture to justify them.

Do the same exercise but use the Chebyshev nodes that are defined in $[-1, 1]$ by

$$x_k = cos\left(\frac{2k - 1}{2n}\pi\right), k = \{1, \dots, n\}.$$

Compute the error when the number of points increases and comment your observations.

Exercise 3. **Linear piecewise interpolation**

Implement a piecewise interpolation for the set of points $x_0, x_1, x_2 = 0,1,2$ and data $\{y_0, y_1, y_2\} = \{1, 3, 2\}$.

Generalize your code for an interval $I = [a, b]$ made of $(n + 1)$ sub-intervals, using a random set of data "numpy.random.rand(N points)".

Apply your code to the sine and Runge functions defined earlier

1. How does the error (use np.linalg.norm) behave as the number of sub-intervals increases?
2. Can you explain it?
3. Mention the differences between the piecewise and global interpolations

Exercise 4 - **Least squares**

Using the monomial polynomial basis, form and solve the normal equations to fit a polynomial of degree n through the data given in Table 1.

t	0.0	1.0	2.0	3.0	4.0	5.0
y	1.0	2.7	5.8	6.6	7.5	9.9

You should write a function SolveLeastSquares in Python that returns the coefficients from solving the normal equations. You can use np.vander, to generate the Vandermonde matrix.

To solve the system you can call the built-in function numpy.linalg.solve.

Compute the found minimum and comment your observations for different values of n.

Print the condition number of the system with numpy.linalg.cond.

Redo the computations by generating a sinusoidal data set with ≈ 200 points, and add some random noise thanks to (for example) numpy.random.normal (Gaussian distribution).

Exercise 5 - **Elliptical orbit**

A planet follows an elliptical orbit, which can be represented in a Cartesian (x, y) coordinate system by the equation

$$ay^2 + bxy + cx + dy + e = x^2$$

Solve the linear least squares to determine the orbital parameters a, b, c, d, e, given the following observations of the planet position

x	1.02	0.95	0.87	0.77	0.67	0.56	0.44	0.30	0.16	0.01
y	0.39	0.32	0.27	0.22	0.18	0.15	0.13	0.12	0.13	0.15

In addition to printing the values for the orbital parameters, plot the resulting orbit and the given data points in the (x, y) plane.